Integración de Reglas de Negocio con Conectores Aspectuales Spring

Graciela Vidal, Juan G. Enriquez y Sandra Casas

Universidad Nacional de la Patagonia Austal,
Unidad Académica Río Gallegos,
Campus Universitario, Av. Gregores y Piloto Lero Rivera
Z9400KVD - Río Gallegos - Argentina
lis@uarg.unpa.edu.ar

Abstract. La Programación Orientada a Aspectos (POA) ha sido propuesta como una alternativa para implementar (encapsular) las conexiones entre reglas de negocio (RN) y la lógica de negocio (funcionalidad base) con el objeto de minimizar las dependencias y acoplamiento y de esta manera minimizar el impacto que tiene la volatilidad de este tipo requisitos en el mantenimiento de las aplicaciones software. Este trabajo plantea como soporte de implementación de los conectores a Spring AOP Framework, un marco de trabajo muy popular en la industria de desarrollo de software. Se presenta el diseño y la implementación de conectores básicos y complejos para integrar RN. También se expone una comparación con los conectores en AspectJ.

Keywords: Programación Orientada a Aspectos, Separación de concerns, Reglas de Negocio, Conexiones aspectuales, Requerimientos volátiles, Spring, AspectJ.

1 Introducción

Las reglas de negocio (RN) reflejan las restricciones que existen en toda organización, de modo que nunca sea posible llevar a cabo acciones no válidas [1]. Las RN son restricciones de tipo legal, político, competitivas, de oportunidad, etc. Una aplicación software implementa un conjunto de RN para dar respuesta acorde a los requerimientos del cliente. Típicamente una RN está representada por una sentencia "if <condición> then <acción>". En las aplicaciones OO suele estar encapsulada en una clase. Los mecanismos habituales de implementación de RN requieren embeber la evaluación de la regla o su invocación en los módulos funcionales o lógica de negocio (funcionalidad base), causando que la implementación se disperse y confunda por múltiples módulos. Un cambio en la especificación de la RN requiere cambios en todos los módulos que están involucrados. Estas modificaciones resultan invasivas y consumen tiempo. Además debido a que las RN son mucho más volátiles que la lógica de negocio, al estar mezcladas causan que los componentes funcionales se tornen también volátiles. Este tipo de implementación de las RN ocasiona inconvenientes para la reutilización y mantenimiento del software.

La Programación Orientada a Aspectos [2] (POA) ha sido propuesta para mejorar la integración de RN con la funcionalidad base [3]. Los enfoques están dirigidos a encapsular las RN y su integración (conector) en los aspectos o ha encapsular sólo el conector a la RN. Estos trabajos se han centrado en proveer soluciones con lenguajes orientados a aspectos de propósito general y de características programáticas como AspectJ [4] y JasCo [5]. Las experiencias indican que aunque se mejora la encapsulación, minimiza la dependencia y favorece la reutilización, aparecen otros inconvenientes. No obstante, los mismos autores indican que es imposible generalizar los resultados a todos los enfoques OA debido a que algunos de ellos son fundamentalmente demasiado diferentes.

Una conextor aspectual entre una RN y la funcionalidad base es todo mecanismo de implementación que permite encapsular: la invocación al objeto que representa a la RN, la obtención y transmisión de la información que la RN requiere, la resolución de la interacción ante la posible simultaneidad de aplicación de más una RN y el retorno de la información generada por la aplicación de la RN. Por mecanismo de implementación se consideran distintos tipos de construcciones como código, anotación, XML u otro elemento que en tiempo de ejecución permita cumplir con una determinada especificación. Este trabajo explora el diseño e implementación de conectores aspectuales de RN con Spring AOP Framework [6].

Las contribuciones principales de este trabajo son: la presentación de lineamientos para el diseño e implementación de conectores aspectuales básicos y complejos con Spring AOP Framework y una comparación preliminar en términos de flexibilidad, reuso y mantenimiento con conectores implementados con AspectJ [4].

La estructura de este trabajo es la siguiente: en la Sección 2 se presenta Spring AOP Framework; en la Sección 3 se presenta el diseño e implementación de los conectores aspectuales con dicha herramienta; en la Sección 4 se expone una comparación con los conectores en AspectJ; y en la Sección 5 se exponen las conclusiones.

2 Spring AOP Framework

Spring es un framework open source que fue creado para abordar el problema de la complejidad de las aplicaciones empresariales [8]. En la actualidad el desarrollo de aplicaciones con Spring se ha masificado, siendo considerado un líder en su tipo. Spring es un contenedor liviano, que ofrece el uso de la técnica llamada inversión de control (IoC) y presenta un módulo de contenedores orientados a aspectos. El soporte POA con Spring presenta cuatro alternativas [9] :

- Spring AOP basados en proxies en tiempo de ejecución (disponible en todas las versiones de Spring)
- Aspectos AspectJ dirigidos por anotaciones (sólo disponible en la versión 2.x de Spring)
- Aspectos POJO puros (sólo disponible en la versión 2.x de Spring)
- Aspectos AspectJ inyectados (disponible en todas las versiones de Spring)

Para el presente trabajo se ha seleccionado el soporte de aspectos POJOS puros, también conocido como soporte de esquemas, por ser el más declarativo y simple en

cuanto a la configuración. En dicho soporte un aspecto es un objeto Java regular, definido como un bean en el Application Context de Spring (XML). El estado y el comportamiento es capturado por campos y métodos del objeto, y los pointcuts y advices son capturados en un configuración AOP XML, delimitada en el Application Context por las etiquetas <aop:config>. El aspecto es declarado usando el elemento <aop:aspect> y el bean es referenciado usando el atributo ref. Este bean puede ser configurado e inyectado como cualquier bean Spring. En la Tabla 1 se sintetizan los elementos para la configuración AOP y sus propósitos

Tabla 1: Elementos para la configuración AOP

Elemento	Propósito
<aop:after>	Define un aviso after
<aop:after-returning>	Define un aviso after-returning
<aop:after-throwing>	Define un aviso after-throwing
<aop:around>	Define un aviso around
<aop:before>	Define un aviso before
<aop:aspect>	Define un aspecto
<aop:config>	Elemento de mayor nivel. Contiene al resto de los elementos
<aop:pointcut>	Define un pointcut

A continuación se aplican estos conceptos con el objeto de encapsular conectores entre RN y eventos de la funcionalidad base.

3 Conectores de RN con aspectos Spring: diseño y configuración

Las reglas del negocio son esenciales para el funcionamiento de las organizaciones [10]. Definen los términos y establecen las políticas centrales del negocio. Controlan o influyen en el comportamiento de la organización ya que establecen que es posible y deseable en la administración de una empresa, y que no lo es. Para los desarrolladores de software, una característica fundamental es su nivel de volatilidad. Esto es inevitablemente así, ya que la dinámica de las organizaciones genera nuevas RN y/o deciden desactivar las existentes, o modificar algún aspecto de las vigentes. Cuando un sistema software se encuentra en etapa de operación, es necesario que este tipo de requerimientos se puedan cumplimntar en forma rápida y lo más simple posible. Por ello, la POA resulta conveniente, al proveer mecanismos que permitan conectar o integrar las RN al dominio sin necesidad de alterar dichos componentes, o como expresa [11] la POA facilita la evolución constante de este tipo concerns. Sin embargo, el diseño e implementación de estos conectores aspectuales, dependen no sólo de las RN sino también de la arquitectura y diseño del sistema al cual deben aplicarse. Aquí es donde puede ocurrir que una RN en un sistema requiera un conector básico y en otro, un conector más complejo.

En la Fig. 1 se presenta un diagrama de la propuesta. La implementación esta dividida en tres grupos de elementos con diferentes propósitos: (a) las clases del dominio, funcionalidad base o lógica de negocio, cuya ejecución ocasiona los eventos (invocación o ejecución de métodos) que activan las RN; (c) el grupo de clases que representan las RN y (b) los conectores aspectuales que se componen de dos

elementos: un bean al que denominaremos aspecto de conexión y la configuración AOP XML (que es parte de la configuración del Application Context en el cual están definidos todos los beans de la aplicación).

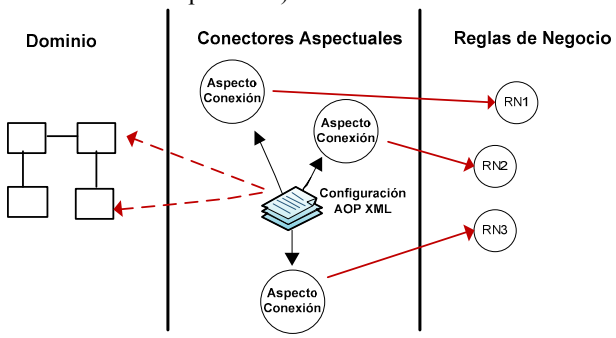

Fig. 1. Diagrama de los elementos con los que interactúan los conectores aspectuales

Para el presente trabajo, elegimos una RN que refleja una política de descuento en los precios para promocionar y aumentar las ventas de un determinado grupo de artículos.

RN#1: Si el artículo consultado es del rubro deporte tiene un 9% de descuento sobre su precio de lista.
Evento: La RN#1 debe ser activada cada vez que se requiera el precio de un artículo de deporte, ya sea para su consulta o para su facturación.

En el esquema, las RN son clases que implementan los métodos *condition()*, *action()* y *apply()* como sugiere el patrón Object Rule [12]. En la Fig. 2 se ha definido al interface BusinessRule y la clase DiscountSport que implementa la RN#1, de acuerdo al patrón.

```
interface BusinessRule {
  boolean condition (..){}
  void action (..) {}
  void apply (..) {}
 }
class DiscountSport implements BusinessRule {
  boolean condition (Item i)
  { return (i.getCategory() == Category.deporte); }
  void action(Item i)
  { return i.setDiscount(9); }
  void apply (Item i) {
    if (condition(i))
       then action(i);
}
```

Fig. 2. Interface BusinessRule y clase DiscountSport

El conector aspectual debe activar la RN cuando la ejecución del programa alcance un determinado evento definido en las clases de negocios. Esto se logra mediante dos elementos: El aspecto de conexión y la configuración AOP XML. El aspecto de conexión, es un bean, que participa de dos configuraciones. Primero este bean es el

responsable de activar a la RN (invocar el método apply), la RN se incluye como una propiedad del bean que será disparada cuando se ejecute el método *triggerBR*. Siguiendo el ejemplo propuesto, el aspecto de conexión será la clase AspectConextionDiscount, como se muestra en la Fig. 3.

```
class  AspectConextionDiscount {
    BusinessRule br;
    void triggerBR(…)
    { br.apply(..)}
}
```

Fig. 3. Estructura básica del aspecto de conexión.

En la configuración del Application Context es necesario inyectar una RN particular al aspecto de conexión. Para esto, se declara la RN como el bean "*brsd*" de tipo SportDiscount, luego al bean "*aspcon*" de tipo AspectConextionDiscount en el atributo "*br*" se asocia el bean "*brsd*", como se describe en la Fig. 4.

```
<beans xmlns = "……..…">
  <bean id="brsd" class="SportDiscount" />
  <bean id="aspcon" class="AspectConextionDiscount">
        <property name="br" ref="brsd" />
  </bean>
  ---
</beans>
```

Fig. 4. Inyección de una RN al aspecto de conexión

Hasta aquí el aspecto de conexión "*aspcon*" ha sido ligado a una RN particular, pero todavía a ningún evento del dominio. Esta segunda configuración, se realiza totalmente en la configuración AOP XML. Se requiere emplear el mismo id de bean, en este caso "*aspcon*" que será etiquetado con <aop_aspect>. Conforme al soporte POA basado en esquemas de Spring, puede hacerse de dos formas, como se presenta en la Fig. 5.

A

```
<aop:config>
  <aop:pointcut id = ”event”
      expression=”execution(Item.getPrice(..))”/>
     <aop:aspect ref=”aspcon”>
        <aop:before
            method="triggerBR”
            pointcut-ref=”event”/>
        </aop:aspect>
 </aop:pointcut>
</aop:config>
```

B

```
<aop:config>
    <aop:aspect ref=”aspcon”>
      <aop:before
          method="triggerBR”
          pointcut= execution (Item.getPrice(..))”/>
    </aop:aspect>
 </aop:config>
```

Fig. 5. Configuraciones AOP XML

En el esquema A, dentro de la configuración AOP XML (<aop:config>) se define el pointcut (<aop:pointcut>) como elemento de mayor nivel, al cual se pueden asociar

uno o más aspectos. Cada aspecto (<aop:aspect>) con un determinado aviso (after, before, around).

En el esquema B, al igual que en el caso anterior, en la configuración AOP XML se define como elemento de nivel mayor el aspecto de conexión mediante la etiqueta <aop:aspect>, a éste se le pueden asociar uno o más avisos y pointcuts.

Un punto de gran variabilidad y vulnerabilidad es el pasaje del contexto, ya que la/s RN requieren datos para que sea posible la evaluación de la condición. Una forma es expresar explícitamente el pasaje del contexto en la definición de los pointcuts, esto seria por ejemplo:

```
<aop:pointcut id = ”event” expression=”execution(Item.getPrice(..)
and this(Item))”/>
```

Este formato reduce las posibilidades de reutilización del pointcut, ya que si en otro caso se requiere otro elemento del contexto, como pueden ser los argumentos del método, la definición del pointcut no resulta útil. Otra posibilidad que ofrece Spring, es que el contexto de ejecución del evento interceptado sea transmitido “implícitamente” al aspecto de conexión y éste lo analice y en consecuencia transmita a la RN los datos del contexto necesarios. En Spring esto es posible, explicitando en el argumento del método del aspecto de conexión que se despacha al interceptar el pointcut, un argumento de clase JoinPoint o ProceedingJoinPoint (el primero para avisos before y after, el segundo para avisos around). Estos objetos permiten obtener información del contexto como, la referencia a this, los argumentos del método interceptado, etc. El método triggerBR que activa la RN en el aspecto de conexión, entonces tendría la forma, que se presenta en la Fig. 6.

```
void triggerBR(JoinPoint jp)
    { br.apply((Item) jp.getThis())}
```

Fig. 6. Método que dispara la RN

Al no ser necesario explicitar el pasaje de contexto en la configuración AOP XML de los pointcuts, se logra que el esquema A (Fig. 5) sea aplicable cuando se requiere asociar más de una RN a un mismo evento. El esquema B (Fig. 5) resultaría aplicable cuando una RN debe ser aplicada a distintos eventos, pero se verá limitada al manejo del contexto que debe realizar el aspecto de conexión.

Otra situación que debe ser resuelta por los conectores aspectuales aparece como consecuencia de la evolución. Los dominios evolucionan cuando nuevas RN son definidas, las cuales se refieren a conceptos del dominio imprevistos en la aplicación actual. Es deseable permitir la definición del nuevo vocabulario del dominio y transparentemente adaptar la actual implementación en orden de aplicar las nuevas RN. Supongamos que se requiere incorporar a una aplicación existente las siguientes nuevas RN:

RN#2: Un cliente es frecuente si tiene más de 20 compras realizadas.

RN#3: Si un cliente es frecuente tiene un descuento del 3% en sus compras.

En la aplicación actual, la clase Costumer no tiene un estado que represente a un cliente como frecuente, lo que implica modificar el dominio para poder luego aplicar la RN#3, referente al descuento. Es decir, aquí existe una dependencia entre las RN, ya que la RN#3 requiere de la clasificación previa realizada por la RN#2.

La RN#3 necesita considerar un estado del dominio que no existe, por lo que el dominio debe ser adaptado. Este tipo de requerimientos desde el enfoque OA se soporta con introducciones (inter-type en AspetcJ). En Spring es posible modificar con aspectos, la estructura estática de una clase, para el caso planteado como se indica en la Fig. 7. Primero, se define una interfaz con el nuevo comportamiento a introducir en el dominio (A). Segundo, se implementa dicha interfaz (B). Tercero, se implementa el aspecto de conexión que dispara la RN#2 que decide sobre el nuevo estado (C), aquí la RN#2 particular es inyectada como se indicó previamente (Fig.4). Por último, se realiza la configuración del aspecto de conexión al dominio, para conectar la RN#2 a un evento particular (D).

```
interface Frecuent {
  boolean isFrecuent{}
  void setFrecuent(boolean f) {}                    A
}
class FrecuentCostumer implements Frecuent {
  boolean frecuent;
  boolean isFrecuent()
  { return frecuent; }                              B
  void setFrecuent(boolean f)
  { fecuent=f; }
}
public class IntroductionFrecuentConextion {
  BusinessRule br;
  public void triggerBR(Costumer costumer)          C
   {  br.apply(costumer);}
}
```

```
<bean id="introduction"
   class="IntroductionFrecuentConextion
   . . . "/>
<aop:config>
  <aop:aspect ref="introduction">
   <aop:declare-parents types-matching="Costumer" implement-
       interface="Frecuent" default-impl="FrecuentCostumer" />      D
   <aop:after
        pointcut="execute(* Invoice.print(..))and args(Costumer c)"
        method="triggerBR" />
  </aop:aspect>
</aop:config>
```

Fig. 7. Adaptación del dominio necesaria para la aplicación de RN.

Para aplicar la RN#3 de descuento especial sobre clientes frecuentes, ahora se debe proceder como se mencionó anteriormente (Fig. 3, 4 y 5), es necesario implementar la RN#3 y el un nuevo aspecto de conexión y se debe definir la configuración del mismo a un evento particular del dominio

Las RN requieren información para evaluar la condición, esta información está en el dominio o es del sistema. Como se observó, a veces es necesario modificar el dominio para luego disponer de la información. Otras veces, la información está en el dominio, pero no en el contexto en el cual se produce el evento que dispara la RN. Por ejemplo, supongamos la siguiente situación:

RN#4: Si la fecha de compra coincide con la fecha de nacimiento del cliente aplicar un descuento del 5%.
Evento: aplicar la RN#4 al calcular el importe total de la factura.

En este caso, se distinguen dos eventos, el evento que dispara la RN#4 y el evento en el cual esta disponible la información que requiere la RN#4. Este segundo evento es cuando se inicializan los detalles de la factura, operación que recibe por parámetro un objeto Costumer, con el propósito de tomar ciertos datos para instanciar estados de la factura como nombre, apellido y domicilio. Este resulta ser el momento adecuado para recuperar la fecha de nacimiento del cliente (que no es un dato requerido para la factura). En la Fig. 8 se presenta un esquema posible, el cual consiste básicamente en que el aspecto de conexión sea empleado en la intercepción de los dos eventos y mantenga la información hasta que deba ser disparada la RN#4.

```
class  AspectConextionDiscount{
  BusinessRule br;
  Date dateOfBirth;;
  void retrieve(Joinpoint jp) {
    dateOfBirth = (Costumer)jp.getThis().getDateOfBirth();
  }
  void triggerBR(Joinpoint jp)
  { br.apply(dateOfBirth);}
}
```

```
<aop:config>
  <aop:aspect ref=”aspcon”>
    <aop:before
       method="retrieve”
       pointcut= execution (* Invoice.setDetails(..))”/>
    <aop:after
       method="triggerBR”
       pointcut= execution (* Invoice.calculateTotal(..))”/>
    </aop:aspect>
</aop:config>
```

Fig. 8. Recuperación de información de otros contextos.

Sin embargo, este esquema tiene dificultades cuando existen múltiples instancias de Invoice, ya que no es posible mantener una instancia del aspecto de conexión por cada una de éstas, siendo una restricción de este tipo de soporte (el modelo de instanciación es singleton). La forma de resolver esta situación, sería entonces la adaptación del dominio (Fig. 7).

4 Conectores Spring vs Conectores AspectJ

AspectJ [4] es el lenguaje orientado a aspectos más popular, difundido y maduro en la actualidad, es una extensión de Java cuyo modelo se ha propagado incluso a otras herramientas POA. Cibrán [7][13] aporta una serie de contribuciones directamente

relacionadas con la problemática de los conectores entre RN y la funcionalidad base en software OO con aspectos cuando analiza el soporte provisto por AspectJ. A continuación realizamos una breve comparación.

La modificación de una RN por otra en AspectJ requiere modificar el código fuente del aspecto que dispara la RN, en Spring esto no es necesario. Mediante la IoC la RN es inyectada al aspecto de conexión, sin necesidad de modificar el código fuente del aspecto; otra posibilidad es modificar en la configuración XML el tipo del bean que corresponde al aspecto de conexión, en este caso tampoco se modifica el código fuente. Esto se debe a que en AspectJ el desarrollador no tiene control en la instanciación e inicialización de los aspectos y en Spring estos aspectos se manejan desde el Application Context. Esta característica de Spring significa una mejora en el mantenimiento de las aplicaciones.

AspectJ permite desacoplar las diferentes partes que constituyen el conector de las RN en aspectos separados para que puedan reutilizarse independientemente. Sin embargo, esto produce una proliferación de aspectos que se tornan difíciles de manejar. En Spring la separación de las distintas partes del aspecto, esta dada en su modelo de esquemas. Los pointcuts se definen en la configuración AOP XML, permitiendo distintas formas de reutilización y las sentencias de ejecución en el aspecto de conexión. Esta característica de Spring permite que la reutilización de los pointcuts no tenga costos adicionales de mantenimiento.

Con AspectJ la reutilización del código del aspecto es posible a través de la herencia, pero los pointcuts de AspectJ son frágiles cuando definen directamente un evento concreto (join-point) y el pasaje del contexto en la ejecución y por consiguiente son menos reusables. En Spring la capacidad de poder manejar el contexto en forma transparente en la definición de los pointcuts y avisos sin lugar a dudas mejora la reutilización de los mismos.

AspectJ es un lenguaje OA estático, cualquier tipo de modificación requiere el código fuente de los aspectos y componentes que éstos interceptan y a su posterior recompilación. Con Spring algunas modificaciones no requieren modificar el código fuente, solo se limitan a modificar la configuración XML, sin necesidades de recompilaciones. Esto se debe a que en AspectJ el tejido es invasivo y Spring emplea la estrategia de proxies.

Si dos o más RN están relacionadas al mismo evento, en AspectJ puede indicarse el orden de ejecución dentro de los aspectos mediante las sentencias “declare precedence”. Cualquier modificación requiere modificar este código fuente y recompilar. Con Spring la precedencia se puede manejar desde la configuración del aspecto de conexión cuando el bean es declarado. Los cambios posteriores en el orden de ejecución no requieren modificación del código fuente ni su recompilación.

Una cuestión que no ha sido estudiada en este trabajo, es el rendimiento en términos de tiempo de ejecución. En este sentido, y de acuerdo a los benchmark presentados en [14] AspectJ tiene una mejor performance que Spring.

5 Conclusiones y Trabajo Futuro

Este trabajo ha presentado algunos lineamientos para conectar RN y lógica de negocio a través de los mecanismos POA, ofrecidos por Spring AOP Framework, específicamente se ha trabajado sobre el soporte de esquemas.

En este sentido Spring presenta algunas características interesantes: el estilo declarativo para integrar eventos y RN, la facilidad de realizar ciertas modificaciones/adaptaciones sin necesidad de manipular el código fuente, la reutilización parcial de los esquemas para los eventos (pointcuts) y RN, la capacidad de asociar aspectos de manera sencilla. En principio estas características han permitido conectar con el soporte POA basado en esquemas requerimientos de distinto nivel de complejidad, de manera flexible. Sin embargo, el modelo de instanciación de aspectos (Singleton) es una restricción importante, particularmente cuando resulta necesario implementar conexiones complejas.

El trabajo futuro esta dirigido a experimentar la implementación de conectores más complejos, y a implementar un mayor número de RN, ya sea con el soporte de esquemas como con el soporte de anotaciones, dado que este último presenta otros modelos de instanciación de aspectos alternativos. Los resultados de los distintos casos planteados, serán empleados para realizar comparaciones entre ambos soportes y con AspectJ.

Agradecimientos

El presente trabajo ha sido parcialmente financiado por la Universidad Nacional de la Patagonia Austral, Santa Cruz, Argentina.

Referencias

1 BRG (2001). “Defining Business Rules: What Are They Really?” Business Rule Group. http://www.businessrulesgroup.org/.

2 Kiczales G., Lamping L., Mendhekar A., Maeda C., Lopes C., Loingtier J., Irwin J., “Aspect-Oriented Programming” (1997). In Proceedings ECOOP’97 – Object-Oriented Programming, 11th European Conference, Finland, Springer-Verlang.

3 Cibrian M., D’Hondt M., & Jonckers V. (2003). “Aspect-oriented programming for connecting business rules”. In Proceedings of the 6th International Conference on Business Information Systems. Colorado, USA.

4 Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. (2001). “An overview of AspectJ”. In Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP ‘01).

5 Vanderperren, W., Suvee, D., Cibrán, M., Verheecke, B. and Jonckers, V. “Adaptive Programming in JAsCo”. (2005) In Proceedings of AOSD, ACM Press, Chicago, USA

6 Sitio Oficial de Spring Framework http://www.springsource.org/

7 Cibrán M. and D’Hondt M. (2003). “Composable and reusable business rules using AspectJ”. In Workshop on Software engineering Properties of Languages for Aspect Technologies (SPLAT) at the International Conference on AOSD. Boston, USA.

8 Johnson R. “Spring Reference 3.0.0.RC1.” 2004-2009.
9 Walls C. and Breidenbach R. (2004) “Spring in Action” Manning Publications Co.
10 Ross R. G. “Principles of the Business Rule Approach”. Addison-Wesley, 2003.
11 Moreira A., Araújo J., and Whittle J. “Modeling Volatile Concerns as Aspects” E. Dubois and K. Pohl (Eds.): CAiSE 2006, LNCS 4001, pp. 544 – 558, 2006.Springer-Verlag Berlin Heidelberg 2006
12 Arsanjani, A. (2001). Rule object 2001: A pattern language for adaptive and scalable business rule construction. Technical Report, IBM T.J. Watson Research Centre.
13 Cibrán M. (2007) “Connecting High-Level Business Rules with Object-Oriented Applications: An approach using Aspect-Oriented Programming and Model-Driven Engineering” Tesis doctoral. Universiteit Brussel.
14 AOP Benchmark. AOP Benchmark - AspectWerkz - Confluence http://docs.codehaus.org/display/AW/AOP+Benchmark